How kernel randomization is canceling memory deduplication in cloud computing systems

Vañó García, Fernando; Marco-Gisbert, Hector

Published in:
2018 IEEE 17th International Symposium on Network Computing and Applications

DOI:
10.1109/NCA.2018.8548338

Published: 01/11/2018

Document Version
Peer reviewed version

How Kernel Randomization is Canceling Memory Deduplication in Cloud Computing Systems

Fernando Vañó-García, Hector Marco-Gisbert
School of Computing, Engineering and Physical Sciences
University of the West of Scotland
High St, Paisley PA1 2BE, UK
{Fernando.Vano-Garcia,Hector.Marco}@uws.ac.uk

***Abstract*—Cloud computing dramatically impacted the way we play, work and live. It has been widely adopted in many sectors mainly because it reduces the cost of performing tasks in a flexible, scalable and reliable way. The highest possible level of protection must be applied in order to provide a secure cloud computing architecture. Unfortunately, the cloud computing paradigm introduces new scenarios where security protection techniques are weakened or disabled to obtain better performance and resources exploitation. An important case is the memory deduplication mechanism which is canceled by the address space layout randomization (ASLR) protection technique.**

In this paper, we present a precise analysis of the impact on the memory deduplication technique when kernel randomization is enabled. Our experiments show that the memory overhead to run 24 kernels is increased by 534% (from 613 MiB to 3.9 GiB) when kernel ASLR is enabled.

***Index Terms*—Cloud, Memory Deduplication, Information Security, KASLR, Memory Management, Virtualization**

I. Introduction

Cloud computing has become a significant aspect of our lives. It allows a provider to share pools of configurable resources (hardware/software) through virtualization, yielding new complex business models that were unpredictable some years ago. Cloud computing has been widely adopted in many sectors, mainly because it reduces the cost of performing tasks in a flexible, scalable and reliable way. From the user's point of view, they can benefit from vast computing power and storage without the need to possess the necessary hardware resources.

Infrastructure as a Service (IaaS) [1] is considered one of the fundamental building blocks for cloud services because, at this level, a client is able to configure virtualized environments with high flexibility without having to concern about deploying large rooms of physical computers. Thence, the service provider supplies the storage, networking and virtualization, so that the client has full control over the system from OS layer upwards. Efficient resource management is fundamental to deal with a proper cloud infrastructure. Hardware resources are a critical asset in the business, and it must be managed and utilized adequately [2]. Given the significance that cloud computing has in people's lives, it is imperative to offer confidentiality, integrity, and availability in any cloud computing architecture.

Although cloud service providers desire to yield as much security as possible in their infrastructure, it is not always possible [3]. Unfortunately, there is no perfect defense mechanism that solves all the problems. Moreover, some security mechanisms can severely affect the efficiency of other performance mechanisms. Kernel randomization is an instance of this issue. When it is enabled along with memory deduplication, the memory saving effectiveness is considerably reduced. As a result, a cloud infrastructure running virtual machines with their kernels being randomized will sustain a forfeit of memory resources.

Throughout this paper, we analyze the impact of kernel randomization in cloud environments and how it is canceling memory deduplication. The remainder of the paper is organized as follows. Section II introduces two background concepts: memory deduplication and kernel randomization. In Section III the problem of memory deduplication being canceled by kernel randomization is presented, and in Section IV the results of our experiments are displayed. Finally, Section V concludes. The main contributions of this paper are the following:

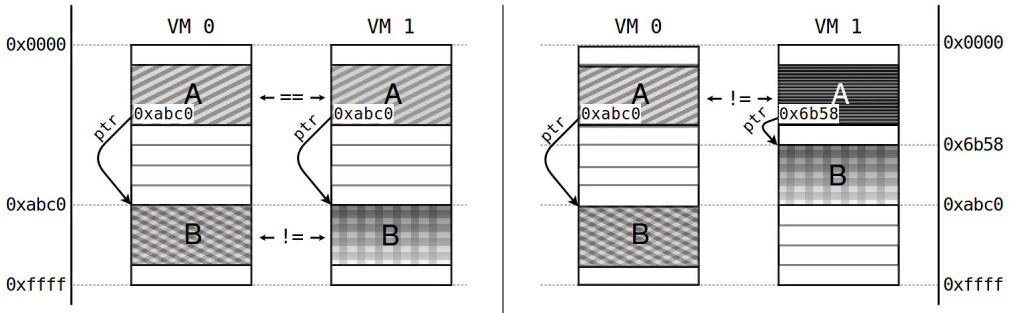
- We detail why the memory deduplication mechanism is not compatible to current kernel Address Space Layout Randomization (KASLR).
- We present an accurate analysis of the Linux kernel memory overhead introduced by the kernel randomization security protection.
- Our experiments in a live cloud environment show that the memory footprint to run only the Linux kernels can be increased by a factor of 6 just by enabling kernel randomization.

II. Background

In this section, we summarize first the memory deduplication memory saving mechanism and later the Kernel Address Space Layout Randomization protection technique.

A. Memory Deduplication

Memory deduplication is a memory saving mechanism that consists in unifying identical pages in order to liberate the space occupied by the redundant copies. Given the noteworthy importance of efficient memory resources utilization on behalf of cloud computing providers, deduplication is a significant feature. It is able to reduce the memory footprint across virtual



(a) Page A is identical between virtual machines, and can be shared by deduplication.

(b) Pointers in pages A differ. Memory deduplication cannot merge any page.

Figure 1: Breakage of memory deduplication by moving memory regions being referenced.

machines [4], [5], decreasing the total cost of managing and ownership.

Pages are compared in pairs. Usually, a hash of the page contents is compared for efficiency reasons. However, at least one bit-to-bit comparison must be performed before merging two identical pages into one single copy in physical memory. This job is typically carried out by a background thread periodically. After a successful merge, both pages will be mapped into a single physical page frame in memory using Copy-On-Write (COW) semantics. As a result, the space occupied by the redundant copy can be released. Subsequently, if another page with the same contents is generated, it can be mapped into the same physical page frame without the need of allocating a new one.

In a virtualized environment, deduplication is commonly applied to the entire memory region corresponding to the virtual machine (often called guest physical memory). Hence, all those pages belonging to that memory region will be candidates for being shared.

B. Kernel Randomization

Address Space Layout Randomization (ASLR) [6] is a security protection technique that consists in placing the memory regions of a program at random locations. The goal of this technique is to hinder the successful exploitation of vulnerabilities (e.g., through buffer overflow) which rely on the knowledge of valid addresses such as code-reuse attacks. As a consequence, an attacker needs to know addresses where code or data is located in memory in order to trigger a malicious payload. It is frequently used in combination with *Data Execution Prevention / No-Execute* (DEP/NX) [7] and the *Stack Smashing Protector* (SSP) [8] resulting in a beneficial complementation [9].

Although ASLR was initially used in userspace programs, it was later applied to the kernel itself. With KASLR, different kernel memory regions are randomized at boot time and will not be changed until next boot. It has been widely adopted by almost all modern operating systems [10]–[12].

III. Memory Deduplication Challenges

Virtualization is a key element used by cloud infrastructure providers, allowing efficient use of physical resources. Physical computers are virtually split into multiple virtual machines, and each virtual machine is delimited to use a subset of the host's resources. Memory deduplication is a significant mechanism, especially in cloud environments where memory resources are a valuable asset, given that it can afford great memory savings. It is applied to pages in the same physical host. Any pair of pages which contents are identical can be merged. An efficient approach to maximize memory sharing by deduplication would be to arrange virtual machines in physical hosts based on similarities in their memory contents, but this is not a simple task, and the effort might be higher than the gains [13]. On the other hand, KASLR or *kernel randomization* is an important mechanism to support system security. Unfortunately, the effectiveness of memory deduplication undergoes a penalty when kernel randomization is present in the guest virtual machines.

The challenge appears when memory deduplication requires pages to contain the same contents in order to merge them, while kernel randomization is changing the contents of guest pages. For example, any memory page containing addresses of internal parts of the kernel will not vary if KASLR is disabled. Contrarily, if KASLR is enabled, those addresses referencing to kernel parts will be adjusted because the kernel memory regions are randomly settled at boot time. Figure 1 represents this issue, where two virtual machines are in the same physical host, and memory deduplication is enabled. Page A contains a pointer to the base address of Page B. In the figure 1a, Page B is located at the same address (`0xabc0`) in both virtual machines. Given that Page A is identical between virtual machines, it can be shared using memory deduplication. This case is equivalent to having two similar virtual machines without kernel randomization. In contrast, in the figure 1b, one of the virtual machines (`VM1`) has moved Page B to a different address (`0x6b58`). As a consequence, the respective pointer in Page A is changed, and the contents of the whole page differ with Page A in the other virtual machine (`VM0`). In this case, it is not possible to share them by deduplication.

Num. VMs	Memory Processed by KSM (MiB)			Memory Saved by KSM (MiB)			Memory Used by VMs (MiB)		
	KASLR Off	KASLR On	Overhead	KASLR Off	KASLR On	Overhead	KASLR Off	KASLR On	Overhead
2	564	584	+3.55%	364	243	-33.24%	200	340	+70.00%
4	1128	1167	+3.46%	888	504	-43.24%	240	663	+176.25%
8	2256	2336	+3.55%	1934	1025	-47.00%	321	1310	+308.10%
16	4513	4673	+3.55%	4046	2069	-48.86%	466	2603	+458.58%
24	6769	7008	+3.53%	6156	3117	-49.37%	613	3891	+534.75%

Table I: Statistics of our experiments, showing the results for 2, 4, 8, 16 and 24 virtual machines running in the host. The data of the first column (Memory Processed by KSM) is the total of the sum of the other two columns: Memory Saved by KSM and Memory Used by the virtual machines.

IV. Analysis and Results

In this section, we present the analysis of memory overhead produced by the Linux kernel randomization mechanism on KSM. Our approach measures pages whose contents depend exclusively on address randomization, in order to calculate the memory overhead produced only by the kernel randomization.

Although there are previous works analyzing the influence of address randomization on memory deduplication [14], those are based on a particular userland workload with a concrete number of virtual machines where both userland and kernel memories were measured. Unfortunately, this approach can not be employed to measure the actual memory overhead produced only by the kernel randomization. Therefore, those results are not valid to properly measure how kernel randomization is canceling memory deduplication, which is necessary to assess and encourage new kernel ASLR designs that have zero or minimum impact on the memory deduplication mechanism.

Userspace activities modify many kernel internal data structures (existing processes, open files, etc.) along with other side effects (e.g., pagecache) that alter the memory contents of the guests, affecting the measurements of memory sharing due to deduplication. However, those userspace activities could vastly variate depending on the particular workload. Consequently, it is possible to select workloads to obtain high memory saving rates but also to select different ones to obtain low rates. Therefore, our experiments are intentionally designed to have no userspace activities, to prevent alterations in the kernel memory from userland (not related with the kernel randomization), in order to accurately measure the memory deduplication cancelations produced by the kernel randomization.

We have compiled the Linux kernel v4.17, as it is the last stable mainline version in kernel.org at the date of writing this paper, with the default configuration for the x86_64 architecture (`x86_64_defconfig`). Thereupon, the experiment consists of running several virtual machines at the same time, enabling and disabling kernel randomization in order to see the differences of memory being saved by KSM. As KASLR support is enabled by default in the chosen configuration, we can disable/enable it with the kernel cmdline, using the flags `nokaslr` and `kaslr` respectively. KSM status information is gathered from the `sysfs` files in `/sys/kernel/mm/ksm/` [15]. The experiments are performed on a machine with an Intel Xeon W-2155 processor running at 3.30 GHz and 32 GiB of SDRAM memory. The hypervisor used is KVM (Linux kernel 4.17.0-ARCH) along with the Qemu emulator version 2.12.0, being managed by Virsh version 4.4.0.

In order to have a minimal userland side in the experiment with negligible impact in our measurements, we have generated a dummy `initrd` image of 411 bytes consisting of a single `init` program that simply calls to `nanosleep()` within an infinite loop.

Linux KSM scans memory regions that have been previously registered for that purpose. Hence, a program needs to mark all those regions whose contents are likely to be shareable. When Qemu launches a virtual machine, it marks the whole region corresponding to the guest physical memory as mergeable. It is important to take into account that the total amount of memory processed by KSM (i.e. potentially shareable memory) does not necessarily correspond with the amount of memory reserved by the host. For efficiency reasons, Linux does not *generate* a page until it is accessed, so KSM will process only pages that the virtual machine has consumed.

Table I shows a summary of the statistics obtained in our experiments. We have launched from 2 to 24 virtual machines at the same time, enabling and disabling KASLR. Each virtual machine has 8 GiB of RAM assigned. In all cases, the amount of memory processed by KSM (which is the total amount of potentially shareable memory) increases approximately a 3.5% when the guest kernels are being randomized. This total

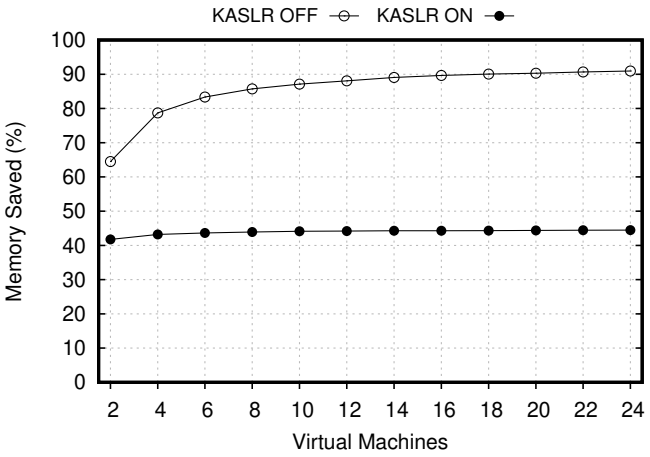


Figure 2: Rate of memory saved by KSM.

amount of memory processed by KSM is divided into two columns: the memory being saved by KSM and the memory used by the virtual machines (i.e., the memory that KSM could not merge). We can observe that enabling KASLR in the guests introduces a memory overhead in both *memory saved* and *memory used* values. The relative overhead of enabling KASLR is shown in the third column. For example, when the host is running 24 virtual machines at the same time, randomizing the kernels introduces an increase in memory used by 534.75% (from 613 MiB to 3891 MiB), and memory deduplication is saving 49.37% less memory.

Figure 2 plots the rate of memory saved, in reference to the total memory processed by KSM. With only two virtual machines, KSM is able to save a 64.5% when kernel randomization is disabled, in contrast with 41.7% when it is enabled. The difference (22.7% of penalty) increases until a certain point (approx. 16 virtual machines). From there, the difference of the memory savings rate by deduplication will be similar, regardless of the number of virtual machines running in the host. With 24 virtual machines, KSM is saving a 90.9% of memory if the kernels are not randomized, in opposition to a 44.4% if kernel randomization is enabled.

From these results, we can separate and quantify the number of pages that are being affected by KASLR. On the one hand, there are pages that do not depend on kernel randomization. When these pages are compared among virtual machines after reaching a stable state, their contents are either always identical (e.g., a page filled with a constant value) or always different (e.g., a page containing cryptographic information). The amount of pages that do not depend on kernel randomization does not vary between the two cases of enabling and disabling kernel randomization. On the other hand, there are pages whose contents depend on kernel randomization. These are pages containing addresses of internal parts of the kernel itself, and they will behave differently in the two cases. If kernel randomization is enabled, those pages will differ among virtual machines, while if kernel randomization is disabled, their contents will be identical. Therefore, the total amount of pages being altered by KASLR can be obtained by subtracting the amount of memory saved when kernel randomization is enabled to the amount of memory saved when kernel randomization is disabled.

In our experiments, when the host is running 24 virtual machines, the 44.89% of the pages are dependant on kernel randomization. These pages are the culprits of the memory sharing cancellation. Their contents embrace addresses related with kernel parts, which vary differently at every boot. Given that their contents differ among virtual machines when kernel randomization is enabled, the memory deduplication mechanism is unable to share them. The memory saving rate decreases almost a 50% compared with the case where the guest kernels are not being randomized. As a consequence, the memory needed to run the kernel of the virtual machines is severely increased.

V. Conclusion and Future Work

Cloud computing relies on multiple technologies to provide cost effective solutions. We discussed that despite the effort and work that is being done in cloud computing technologies there are still challenging problems to combine them with security solutions.

In this paper we analyzed the memory deduplication optimizer present in all modern hypervisors and how it is canceled by the kernel randomization protection mechanism. Our experiments have been designed to accurately measure the memory deduplication cancellations produced by the kernel randomization. They revealed that 3278 MiB of additional memory (+534.75%) are used to run 24 kernel guests.

This work opens the way to new lines of research on having new kernel ASLR designs that have zero or minimum impact on the memory deduplication mechanism.

References

[1] P. Mell, T. Grance *et al.*, "The nist definition of cloud computing," 2011.

[2] B. Sotomayor, R. S. Montero, I. M. Llorente, and I. Foster, "Virtual infrastructure management in private and hybrid clouds," *IEEE Internet computing*, vol. 13, no. 5, 2009.

[3] B. R. Kandukuri, A. Rakshit *et al.*, "Cloud security issues," in *Services Computing, 2009. SCC'09. IEEE International Conference on*. IEEE, 2009, pp. 517–520.

[4] D. Gupta, S. Lee, M. Vrable, S. Savage, A. C. Snoeren, G. Varghese, G. M. Voelker, and A. Vahdat, "Difference engine: Harnessing memory redundancy in virtual machines," in *Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'08. Berkeley, CA, USA: USENIX Association, 2008, pp. 309–322. [Online]. Available: http://dl.acm.org/citation.cfm?id=1855741.1855763

[5] A. Arcangeli, I. Eidus, and C. Wright, "Increasing memory density by using ksm," in *In OLS*, 2009.

[6] Pax address space layout randomization (aslr). [Retrieved: Sep, 2018]. [Online]. Available: https://pax.grsecurity.net/docs/aslr.txt

[7] T. Krazit, "Pcworld-news-amd chips guard against trojan horses," *IDG News Service*, 2004.

[8] H. Etoh, "Stack smashing protector (ssp)," 2005.

[9] H. M. Gisbert and I. Ripoll, "On the effectiveness of nx, ssp, renewssp, and aslr against stack buffer overflows," in *2014 IEEE 13th International Symposium on Network Computing and Applications*, Aug 2014, pp. 145–152.

[10] Z. Xu, G. Liu, T. Wang, and H. Xu, "Exploitations of uninitialized uses on macos sierra," in *11th USENIX Workshop on Offensive Technologies (WOOT 17)*. Vancouver, BC: USENIX Association, 2017. [Online]. Available: https://www.usenix.org/conference/woot17/workshop-program/presentation/xu

[11] "Address space layout randomization in windows vista," May 2006, [Retrieved: Sep, 2018]. [Online]. Available: https://blogs.msdn.microsoft.com/michael_howard/2006/05/26/address-space-layout-randomization-in-windows-vista/

[12] "Linux 3.16: enable config_randomize_base," Aug 2014, [Retrieved: Sep, 2018]. [Online]. Available: bugs.archlinux.org/task/41463

[13] S. Rampersaud and D. Grosu, "A sharing-aware greedy algorithm for virtual machine maximization," in *2014 IEEE 13th International Symposium on Network Computing and Applications*, Aug 2014, pp. 113–120.

[14] K. Suzaki, K. Iijima, T. Yagi, and C. Artho, "Effects of memory randomization, sanitization and page cache on memory deduplication," in *Proc. European Workshop on System Security (EuroSec 2012) :*, 2012, qC 20170104.

[15] "Linux kernel documentation: How to use the kernel samepage merging feature," [Retrieved: Sep, 2018]. [Online]. Available: https://www.kernel.org/doc/Documentation/vm/ksm.txt